

QuanTree and QuanLin, Two Special Purpose Quantum Compilers

Robert R. Tucci

P.O. Box 226

Bedford, MA 01730

tucci@ar-tiste.com

February 17, 2008

Abstract

This paper introduces QuanTree v1.1 and QuanLin v1.1, two Java applications available for free. (Source code included in the distribution.) Each application compiles a different type of input quantum evolution operator. The applications output a quantum circuit that is approximately equal to the input evolution operator. QuanTree compiles an input evolution operator whose Hamiltonian is proportional to the incidence matrix of a balanced, binary tree graph. QuanLin compiles an input evolution operator whose Hamiltonian is proportional to the incidence matrix of a line (open string) graph. Both applications also output an error, defined as the distance in the Frobenius norm between the input evolution operator and the output quantum circuit.

1 Introduction

We say a unitary operator acting on a set of qubits has been compiled if it has been expressed as a SEO (sequence of elementary operations, like CNOTs and single-qubit operations). SEO's are often represented as quantum circuits.

There exist software (quantum compilers) like Qubiter[1] for compiling arbitrary unitary operators (operators that have no a priori known structure). This paper introduces two special purpose quantum compilers, QuanTree and QuanLin. They are special purpose in the sense that they can only compile unitary operators that have a very definite, special structure.

QuanTree v1.0 and QuanLin v1.1 are two Java applications, available[2] for free. (Source code included in the distribution.) Each application compiles a different kind of input quantum evolution operator. The applications output a quantum circuit that is approximately equal to the input evolution operator. QuanTree compiles an input evolution operator whose Hamiltonian is proportional to the incidence matrix of a balanced, binary tree graph. QuanLin compiles an input evolution operator whose Hamiltonian is proportional to the incidence matrix of a line (open string) graph. Both applications also output an error, defined as the distance in the Frobenius norm between the input evolution operator and the output quantum circuit.

Recently, Farhi-Goldstone-Gutmann (FGG) wrote a paper[3] that proposes a quantum algorithm for evaluating NAND formulas via a quantum walk over a tree graph connected to a line (“runway”) graph. Their paper has inspired a flurry of papers expanding on their ideas. Among these papers is one[4] written by me, which provides all the theoretical underpinnings of QuanTree and QuanLin. Please refer to Ref.[4] and the source code of QuanTree and QuanLin if you have any technical questions that are not addressed here. To get a quantum circuit for the FGG algorithm requires first finding a quantum circuit for the evolution operators of a tree and line graph, which is what QuanTree and QuanLin do. A future paper will combine QuanTree and QuanLin software to give a quantum circuit for the full FGG algorithm.

The standard definition of the evolution operator in Quantum Mechanics is $U = e^{-itH}$, where t is time and H is a Hamiltonian. Throughout this paper, we will set $t = -1$ so $U = e^{iH}$. If H is proportional to a coupling constant g , reference to time can be restored easily by replacing the symbol g by $-tg$, and the symbol H by $-tH$.

2 QuanTree

2.1 Input Evolution Operator

A binary tree with $\Lambda + 1$ levels has $1 + 2 + 2^2 + \dots + 2^\Lambda = 2^{\Lambda+1} - 1$ nodes (a.k.a. as states). To reach $N_S = 2^{\Lambda+1}$ states, we add an extra “dead” or “dud” node, labelled with the letter “ d ”. This d node is not connected to any other node in the graph. If we include this dud node, then the number of leaves N_{lvs} is exactly half the number of nodes: $N_{lvs} = \frac{1}{2}2^{\Lambda+1} = 2^\Lambda$. We will often use N_B for the number of bits and $N_S = 2^{N_B}$ for the number of states. Therefore, $N_B = \Lambda + 1$. For example, Fig.1 shows a binary-tree graph with $\Lambda = 2$ and $N_B = 3$. It has $N_S = 2^{\Lambda+1}=8$ nodes, labelled $d, 1, 2, \dots, 7$, half of which (4, 5, 6, 7) are leaves.

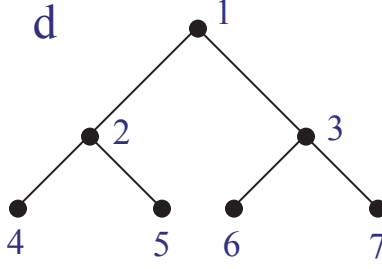


Figure 1: Binary tree with 8 nodes

The Hamiltonian for transitions along the edges of the graph Fig.1 is:

$$H_{tree} = g \begin{array}{c|cccccccc} & d & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \hline d & & & & & & & & \\ \hline 1 & & & 1 & 1 & & & & \\ \hline 2 & & 1 & & & 1 & 1 & & \\ \hline 3 & & 1 & & & & & 1 & 1 \\ \hline 4 & & & 1 & & & & & \\ \hline 5 & & & 1 & & & & & \\ \hline 6 & & & & 1 & & & & \\ \hline 7 & & & & 1 & & & & \end{array}, \quad (1)$$

where g is a real number that we will call the **coupling constant**. In Eq.(1), empty matrix entries represent zero.

For $N_B = 3$ qubits (i.e., $2^{N_B} = 8$ states), the **input evolution operator** for QuanTree is $U = e^{iH_{tree}}$, where H_{tree} is given by Eq.(1). It is easy to generalize Fig.1 and Eq.(1) to arbitrary N_B . QuanTree can compile $e^{iH_{tree}}$ for $N_B \in \{2, 3, 4, \dots\}$.

For $r = 1, 2, 3, \dots$, if $U = L_r(g) + \mathcal{O}(g^{r+1})$, we say $L_r(g)$ **approximates (or is an approximant) of order r** for U .

Given an approximant $L_r(g) + \mathcal{O}(g^{r+1})$ of U , and some $N_T = 1, 2, 3, \dots$, one can approximate U by $\left(L_r\left(\frac{g}{N_T}\right)\right)^{N_T} + \mathcal{O}\left(\frac{g^{r+1}}{N_T^r}\right)$. We will refer to this as Trotter's trick, and to N_T as the **number of trots**.

For $N_T = 1$, QuanTree approximates $e^{iH_{tree}}$ with an approximant of order 3 that is derived in Ref.[4]. Thus, for $N_T = 1$, the error is $\mathcal{O}(g^4)$. For $N_T > 1$, the error is $\mathcal{O}\left(\frac{g^4}{N_T^3}\right)$.

2.2 The Control Panel

Fig.2 shows the **Control Panel** for QuanTree. This is the main and only window of the application. This window is open if and only if the application is running.



Figure 2: Control Panel of QuanTree

The Control Panel allows you to enter the following inputs:

File Prefix: Prefix to the 3 output files that are written when you press the **Write Files** button. For example, if you insert **test** in this text field, the following 3 files will be written:

- `test_qtree_log.txt` (See Fig.3)
- `test_qtree_eng.txt` (See Fig.4)
- `test_qtree_pic.txt` (See Fig.5)

Number of Qubits: The parameter $N_B = 2, 3, 4, \dots$ defined above.

Coupling Constant: The parameter $g \in \mathbb{R}$ defined above.

Number of Trots: The parameter $N_T = 1, 2, 3, \dots$ defined above.

The Control Panel displays the following outputs:

Number of Elementary Operations: The number of elementary operations in the output quantum circuit. If there are no LOOPS, this is the number of lines in the English File (see Sec. 2.3.2), which equals the number of lines in the Picture File (see Sec. 2.3.3). When there are LOOPS, the “LOOP k REPS: N_T ” and “NEXT k” lines are not counted, whereas the lines between “LOOP k REPS: N_T ” and “NEXT k” are counted N_T times.

Error: The distance in the Frobenius norm between the input evolution operator and the output quantum circuit (i.e., the SEO given in the English File). For a nice review of matrix norms, see Ref.[5]. For any matrix $A \in \mathbb{C}^{n \times n}$, its Frobenius norm is defined as $\|A\|_F = \sqrt{\sum_{j,k} A_{j,k} A_{j,k}^*}$. Another common matrix norm is the 2-norm. The 2-norm $\|A\|_2$ of A equals the largest singular value of A . The Frobenius and 2-norm of A are related by[5]: $\|A\|_2 \leq \|A\|_F \leq \sqrt{2}\|A\|_2$. Since the approximant used by QuanTree is of order 3, if ϵ denotes the error, then $\epsilon(g) \approx K g^4$, for some $K \in \mathbb{R}$ and $|g| < 1$. Thus,

$$\frac{\log(\epsilon(g_2)/\epsilon(g_1))}{\log(g_2/g_1)} \approx 4. \quad (2)$$

For example, with $N_B = 4$ and $N_T = 1$, QuanTree gives $\epsilon(0.05) = 1.383 \times 10^{-5}$ and $\epsilon(0.06) = 2.923 \times 10^{-5}$, which gives $\log(\epsilon_2/\epsilon_1)/\log(g_2/g_1) = 4.11$.

Message: A message appears in this text field if you press **Write Files** with a bad input. The message tries to explain the mistake in the input.

2.3 Output Files

Figs.3, 4, 5, were all generated in a single run of QuanTree (by pressing the **Write Files** button just once). They are examples of what we call the **Log File**, **English File**, and **Picture File**, respectively, of QuanTree. Next we explain the contents of each of these output files.

2.3.1 Log File

Fig.3 is an example a Log File. The Log File records all the information found in the Control Panel.

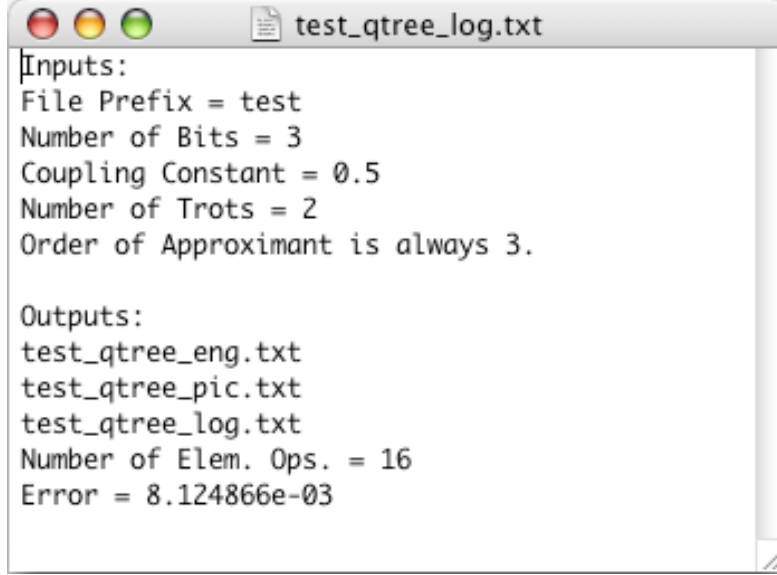


Figure 3: Log File generated by QuanTree

2.3.2 English File

Fig.4 is an example of an English File. The English File completely specifies the output SEO. It does so “in English”, thus its name. Each line represents one elementary operation, and time increases as we move downwards.

In general, an English File obeys the following rules:

- Time grows as we move down the file.
- Each row corresponds to one elementary operation. Each row starts with 4 letters that indicate the type of elementary operation.
- For a one-bit operation acting on a “target bit” α , the target bit α is given after the word AT.
- If the one-bit operation is controlled, then the controls are indicated after the word IF. T and F stand for true and false, respectively. α T stands for a control $n(\alpha)$ at bit α . α F stands for a control $\bar{n}(\alpha)$ at bit α .
- “LOOP k REPS: N_T ” and “NEXT k” mark the beginning and end of N_T Trotter iterations. k labels the loop. k also equals the line-count number (first line is 0) of the line “LOOP k REPS: N_T ” in the English file.
- SWAP $\alpha \beta$ stands for the swap(exchange) operator $E(\alpha, \beta)$ that swaps bits α and β .

```

test_qtree_eng.txt
ROTY -90.0 AT 0 IF 2T
SWAP 1 0 IF 2T
ROTY -90.0 AT 0 IF 1T
LOOP 3 REPS: 2
ROTX 20.257117113534886 AT 1 IF 2F
ROTX 40.30355205094235 AT 2 IF 1T 0T
ROTX -20.257117113534886 AT 1 IF 2F
ROTX 40.93625750026842 AT 1 IF 2F 0T
ROTX 40.51423422706978 AT 2 IF 1T 0F
NEXT 3
ROTY 90.0 AT 0 IF 1T
SWAP 1 0 IF 2T
ROTY 90.0 AT 0 IF 2T

```

Figure 4: English File generated by QuanTree

- PHAS stands for a controlled one-bit gate, where the one-bit gate consists of $diag(1, 1)$ times an angle (“phase”).
- POPH stands for a controlled one-bit gate, where the one-bit gate consists of $P_0 = \bar{n}()$ times an angle (“phase”). P1PH stands for a controlled one-bit gate, where the one-bit gate consists of $P_1 = n()$ times an angle (“phase”).
- SIGX, SIGY, SIGZ, HAD2 stand for the Pauli matrices $\sigma_X, \sigma_Y, \sigma_Z$ and the one-bit Hadamard matrix H .
- ROTX, ROTY, ROTZ, ROTN stand for rotations with rotation axes in the directions: x, y, z , and an arbitrary direction n , respectively.

Here is a list of examples showing how to translate the mathematical notation used in Ref.[4] into the English File language:

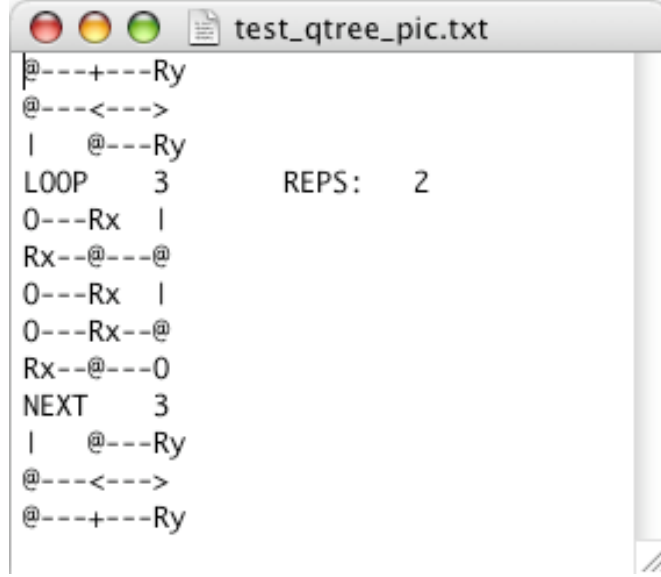


Figure 5: Picture File generated by QuanTree

Mathematical language	English File language
Loop called 5 with 2 repetitions	LOOP 5 REPS: 2
Next iteration of loop called 5	NEXT 5
$E(1,0)\bar{n}(3)n(2)$	SWAP 1 0 IF 3F 2T
$e^{i42.7\bar{n}(3)n(2)}$	PHAS 42.7 IF 3F 2T
$e^{i42.7\bar{n}(3)n(2)}$	POPH 42.7 AT 3 IF 2T
$e^{i42.7n(3)n(2)}$	P1PH 42.7 AT 3 IF 2T
$\sigma_X(1)\bar{n}(3)n(2)$	SIGX AT 1 IF 3F 2T
$\sigma_Y(1)\bar{n}(3)n(2)$	SIGY AT 1 IF 3F 2T
$\sigma_Z(1)\bar{n}(3)n(2)$	SIGZ AT 1 IF 3F 2T
$H(1)\bar{n}(3)n(2)$	HAD2 AT 1 IF 3F 2T
$(e^{\frac{i}{2}\frac{\pi}{180}23.7\sigma_X(1)})\bar{n}(3)n(2)$	ROTX 23.7 AT 1 IF 3F 2T
$(e^{\frac{i}{2}\frac{\pi}{180}23.7\sigma_Y(1)})\bar{n}(3)n(2)$	ROTY 23.7 AT 1 IF 3F 2T
$(e^{\frac{i}{2}\frac{\pi}{180}23.7\sigma_Z(1)})\bar{n}(3)n(2)$	ROTZ 23.7 AT 1 IF 3F 2T
$(e^{\frac{i}{2}\frac{\pi}{180}[30\sigma_X(1)+40\sigma_Y(1)+11\sigma_Z(1)]})\bar{n}(3)n(2)$	ROTN 30.0 40.0 11.0 AT 1 IF 3F 2T

2.3.3 ASCII Picture File

Fig.5 is an example of a Picture File. The Picture File partially specifies the output SEO. It gives an ASCII picture of the quantum circuit. Each line represents one elementary operation, and time increases as we move downwards. There is a one-to-one correspondence between the rows of the English and Picture Files.

In general, a Picture File obeys the following rules:

- Time grows as we move down the file.
- Each row corresponds to one elementary operation. Columns 1, 5, 9, 13, ... represent qubits (or, qubit positions). We define the rightmost qubit as 0. The qubit immediately to the left of the rightmost qubit is 1, etc. For a one-bit operator acting on a “target bit” α , one places a symbol of the operator at bit position α .
- | represents a wire connecting the same qubit at two times.
- - represents a wire connecting different qubits at the same time.
- + represents both | and -.
- If the one-bit operation is controlled, then the controls are indicated as follows. @ at bit position α stands for a control $n(\alpha)$. 0 at bit position α stands for a control $\bar{n}(\alpha)$.
- “LOOP k REPS: N_T ” and “NEXT k ” mark the beginning and end of N_T Trotter iterations. k labels the loop. k also equals the line-count number (first line is 0) of the line “LOOP k REPS: N_T ” in the picture file.
- The swap(exchange) operator $E(\alpha, \beta)$ is represented by putting arrow heads < and > at bit positions α and β .
- A phase factor $e^{i\theta}$ for some angle θ is represented by placing Ph at any bit position which does not already hold a control.
- The one-bit gate $P_0(\alpha) = \bar{n}(\alpha)$ times an angle is represented by putting OP at bit position α .
- The one-bit gate $P_1(\alpha) = n(\alpha)$ times an angle is represented by putting @P at bit position α .
- One-bit operations $\sigma_X(\alpha)$, $\sigma_Y(\alpha)$, $\sigma_Z(\alpha)$ and $H(\alpha)$ are represented by placing the letters X, Y, Z, H, respectively, at bit position α .

- One-bit rotations acting on bit α , in the x, y, z, n directions, are represented by placing **Rx, Ry, Rz, R**, respectively, at bit position α .

Here is a list of examples showing how to translate the mathematical notation used in Ref.[4] into the Picture File language:

Mathematical language	Picture File language
Loop called 5 with 2 repetitions	LOOP 5 REPS:2
Next iteration of loop called 5	NEXT 5
$E(1, 0)^{\bar{n}(3)n(2)}$	0---@---<--->
$e^{i42.7\bar{n}(3)n(2)}$	0---@---+---Ph
$e^{i42.7\bar{n}(3)n(2)}$	0P--@
$e^{i42.7n(3)n(2)}$	@P--@
$\sigma_X(1)^{\bar{n}(3)n(2)}$	0---@---X
$\sigma_Y(1)^{\bar{n}(3)n(2)}$	0---@---Y
$\sigma_Z(1)^{\bar{n}(3)n(2)}$	0---@---Z
$H(1)^{\bar{n}(3)n(2)}$	0---@---H
$(e^{\frac{i}{2} \frac{\pi}{180} 23.7\sigma_X(1)})^{\bar{n}(3)n(2)}$	0---@---Rx
$(e^{\frac{i}{2} \frac{\pi}{180} 23.7\sigma_Y(1)})^{\bar{n}(3)n(2)}$	0---@---Ry
$(e^{\frac{i}{2} \frac{\pi}{180} 23.7\sigma_Z(1)})^{\bar{n}(3)n(2)}$	0---@---Rz
$(e^{\frac{i}{2} \frac{\pi}{180} [30\sigma_X(1)+40\sigma_Y(1)+11\sigma_Z(1)]})^{\bar{n}(3)n(2)}$	0---@---R

2.4 Behind the Scenes

Brief summary of the steps taken by QuanTree every time you press the **Write Files** button:

1. Generate the English and Picture Files according to the rules of Ref.[4].
2. Generate H . Calculate the eigenvalues and eigenvectors of H . Use this eigen-system to calculate $U = e^{iH}$.
3. Read the English File that was written in Step 1. Multiply out the SEO given by the English File to obtain a unitary matrix U' . Calculate the error $\|U - U'\|_F$.
4. Generate the Log File.

3 QuanLin

3.1 Input Evolution Operator

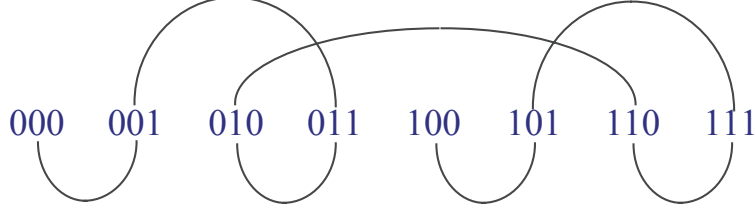


Figure 6: Line (open string) with 8 nodes

Let N_B be the number of bits and $N_S = 2^{N_B}$ the number of states. Fig.6 shows the 8 possible states for 3 bits. The states read from left to right are in increasing “decimal ordering”. These 8 states can also be ordered in “Gray code ordering”. A sequence of Gray code is one wherein two consecutive states are labelled by a binary number and these labels differ only at one bit position. In Fig.6, states connected by an edge (curved line) are consecutive in a Gray code ordering.

The Hamiltonian for transitions along the edges of the graph Fig.6 is:

$$H_{line} = g \begin{array}{c|cccccccc} & 000 & 001 & 010 & 011 & 100 & 101 & 110 & 111 \\ \hline 000 & 0 & 1 & & & & & & \\ 001 & 1 & 0 & & 1 & & & & \\ 010 & & & 0 & 1 & & & 1 & \\ 011 & & 1 & 1 & 0 & & & & \\ 100 & & & & & 0 & 1 & & \\ 101 & & & & & 1 & 0 & & 1 \\ 110 & & & 1 & & & & 0 & 1 \\ 111 & & & & & & 1 & 1 & 0 \end{array}, \quad (3)$$

where g is a real number that we will call the **coupling constant**. In Eq.(3), empty matrix entries represent zero.

For $N_B = 3$ qubits (i.e., $2^{N_B} = 8$ states), the **input evolution operator** for QuanLin is $U = e^{iH_{line}}$, where H_{line} is given by Eq.(3). It is easy to generalize Fig.6 and Eq.(3) to arbitrary N_B . QuanLin can compile $e^{iH_{line}}$ for $N_B \in \{2, 3, 4, \dots\}$.

For $r = 1, 2, 3, \dots$, if $U = L_r(g) + \mathcal{O}(g^{r+1})$, we say $L_r(g)$ **approximates (or is an approximant) of order r for U** .

Given an approximant $L_r(g) + \mathcal{O}(g^{r+1})$ of U , and some $N_T = 1, 2, 3 \dots$, one can approximate U by $\left(L_r(\frac{g}{N_T})\right)^{N_T} + \mathcal{O}(\frac{g^{r+1}}{N_T^r})$. We will refer to this as Trotter's trick, and to N_T as the **number of trots**.

For $N_T = 1$, QuanLin approximates $e^{iH_{line}}$ with a Suzuki approximant of order $r = 2, 4, 6, \dots$ that is derived in Ref.[4]. Thus, for $N_T = 1$, the error is $\mathcal{O}(g^{r+1})$. For $N_T > 1$, the error is $\mathcal{O}(\frac{g^{r+1}}{N_T^r})$.

3.2 The Control Panel

Fig.7 shows the **Control Panel** for QuanLin. This is the main and only window of the application. This window is open if and only if the application is running.

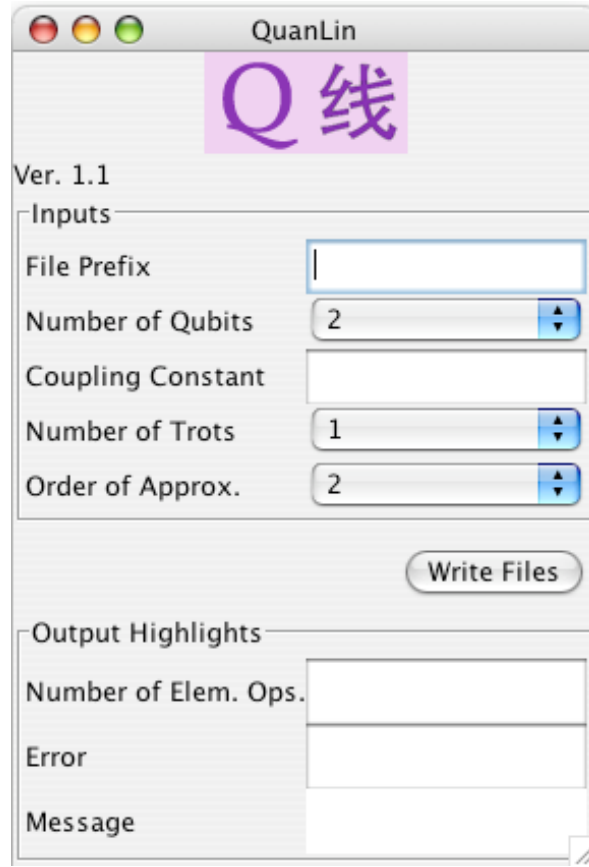
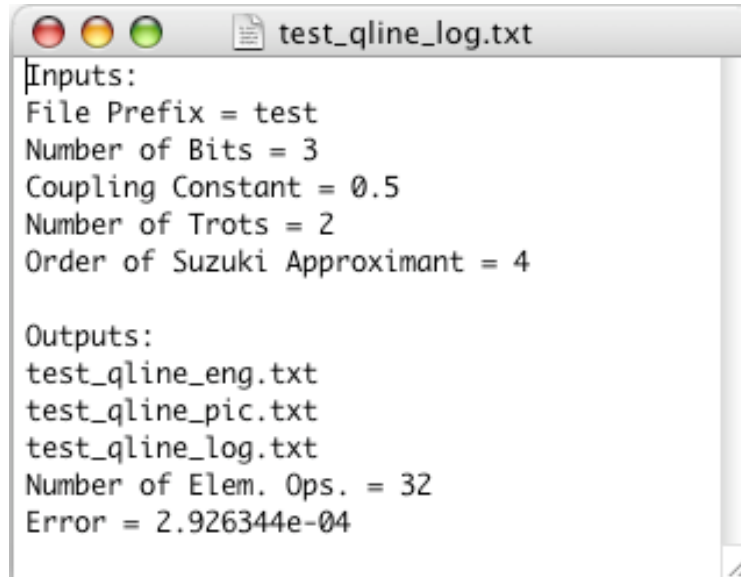


Figure 7: Control Panel of QuanLin

The Control Panel for QuanLin is almost identical to that for QuanTree. The significance of the various data fields in the Control Panel for QuanLin is the same as for QuanTree.

3.3 Output Files

Figs.8, 9, 10, were all generated in a single run of QuanLin (by pressing the **Write Files** button just once). They are examples of what we call the **Log File**, **English File**, and **Picture File**, respectively, of QuanLin. These files are analogous to their namesakes for QuanTree. They follow the same rules.



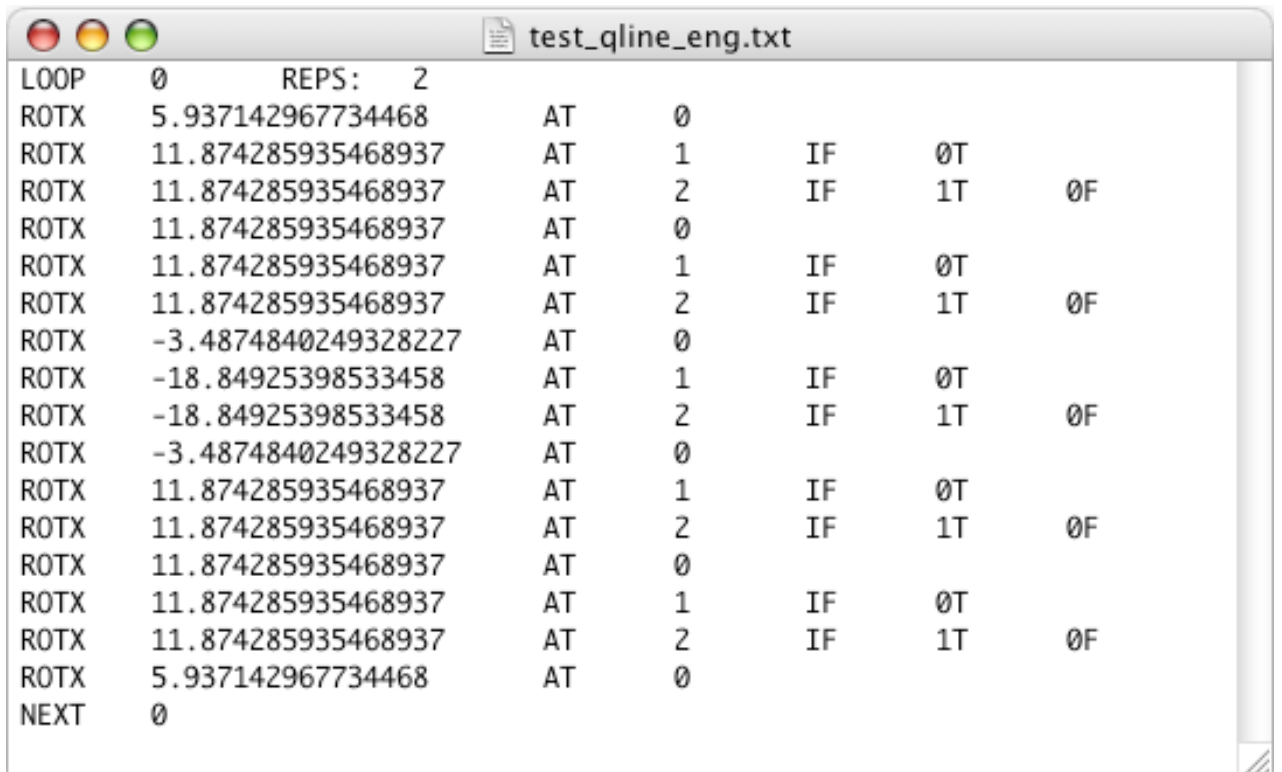
```
test_qline_log.txt
Inputs:
File Prefix = test
Number of Bits = 3
Coupling Constant = 0.5
Number of Trots = 2
Order of Suzuki Approximant = 4

Outputs:
test_qline_eng.txt
test_qline_pic.txt
test_qline_log.txt
Number of Elem. Ops. = 32
Error = 2.926344e-04
```

Figure 8: Log File generated by QuanLin

References

- [1] R.R. Tucci, “A Rudimentary Quantum Compiler(2cnd Ed.)”, arXiv:quant-ph/9902062 . Qubiter software available at www.ar-tiste.com/qubiter.html
- [2] QuanTree and QuanLin software available at www.ar-tiste.com/QuanSuite.html
- [3] E. Farhi, J. Goldstone, S. Gutmann, “A Quantum Algorithm for the Hamiltonian NAND Tree”, arXiv:quant-ph/0702144
- [4] R.R.Tucci, “How to Compile Some NAND Formula Evaluators”, arXiv:0706.0479
- [5] G.H. Golub and C.F. Van Loan, *Matrix Computations, Third Edition* (John Hopkins Univ. Press, 1996).



```
test_qline_eng.txt
LOOP      0      REPS:  2
ROTX      5.937142967734468      AT      0
ROTX      11.874285935468937      AT      1      IF      0T
ROTX      11.874285935468937      AT      2      IF      1T      0F
ROTX      11.874285935468937      AT      0
ROTX      11.874285935468937      AT      1      IF      0T
ROTX      11.874285935468937      AT      2      IF      1T      0F
ROTX      -3.4874840249328227      AT      0
ROTX      -18.84925398533458      AT      1      IF      0T
ROTX      -18.84925398533458      AT      2      IF      1T      0F
ROTX      -3.4874840249328227      AT      0
ROTX      11.874285935468937      AT      1      IF      0T
ROTX      11.874285935468937      AT      2      IF      1T      0F
ROTX      11.874285935468937      AT      0
ROTX      11.874285935468937      AT      1      IF      0T
ROTX      11.874285935468937      AT      2      IF      1T      0F
ROTX      5.937142967734468      AT      0
NEXT      0
```

Figure 9: English File generated by QuanLin

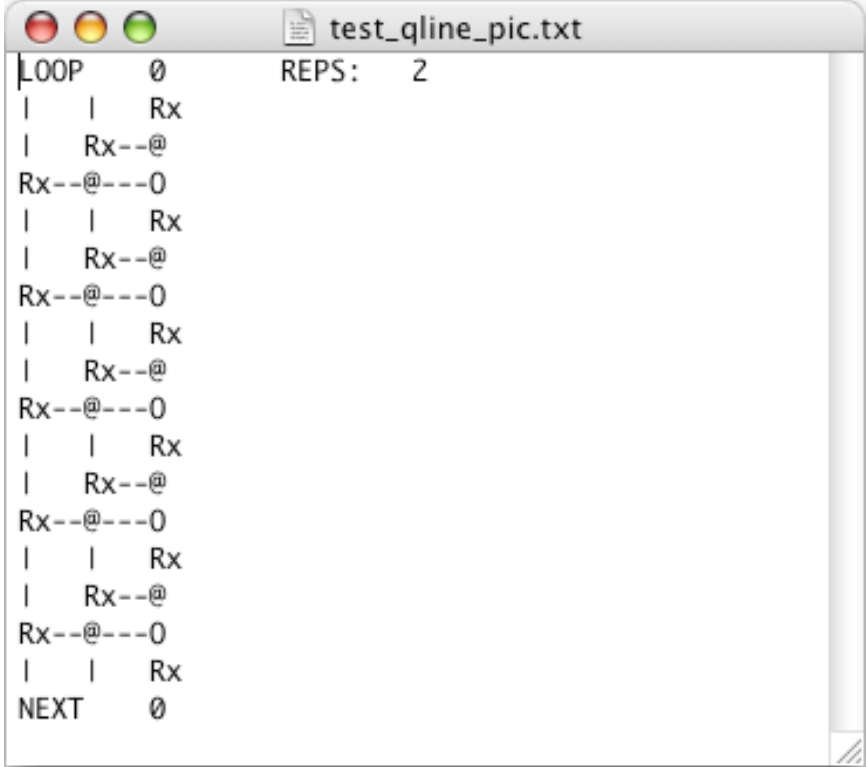


Figure 10: Picture File generated by QuanLin